



**Eur päisches
Patentamt**

**Eur pean
Patent Office**

**Office européen
des brevets**

Bescheinigung

Certificate

Attestation

Die angehefteten Unterla-
gen stimmen mit der
ursprünglich eingereichten
Fassung der auf dem näch-
sten Blatt bezeichneten
europäischen Patentanmel-
dung überein.

The attached documents
are exact copies of the
European patent application
described on the following
page, as originally filed.

Les documents fixés à
cette attestation sont
conformes à la version
initialement déposée de
la demande de brevet
européen spécifiée à la
page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

02425437.7

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk



Anmeldung Nr:
Application no.: 02425437.7
Demande no:

Anmeldetag:
Date of filing: 02.07.02
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

STMicroelectronics S.r.l.
Via C. Olivetti, 2
20041 Agrate Brianza (Milano)
ITALIE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s)
revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F9/00

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of
filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR IE IT LI LU MC NL PT SE SK TR

"Procedimento per eseguire programmi con
processori con lunghezza di istruzione selezionabile e
relativo sistema processore"

5

Campo dell'invenzione

La presente invenzione si riferisce ai processori
ed ai procedimenti per l'elaborazione dei segnali
10 attuabili con gli stessi.

Descrizione della tecnica nota

Nei sistemi per telefonia cellulare di seconda
generazione (ad esempio GSM) o di tipo più evoluto
(GPRS, EDGE, UMTS), l'architettura più frequentemente
15 utilizzata consiste di un sistema costituito da due
processori. Il primo processore, specializzato nel
gestire la parte con più rilevante onere
computazionale, è tipicamente costituito da un Digital
Signal Processor o DSP. L'altro processore, con compiti
20 di controllo, sincronizzazione ed esecuzione di
applicazioni ad alto livello, si configura tipicamente
come una CPU.

Un esempio di architettura di questo genere è
rappresentata nella figura 1, dove i suddetti
25 processori, indicati rispettivamente come DSP e CPU 1,
sono rappresentati insieme alle memorie cache ad essi
associate, rispettivamente in funzione di cache
istruzioni I\$ e di cache dati D\$.

Con CMC sono indicati i moduli di interfaccia,
30 denominati Core Memory Controller, che permettono ai
due sotto - sistemi facenti capo ai due processori DSP
e CPU 1 di interfacciarsi tramite un bus principale B
con la memoria principale di sistema MEM e con le varie
unità periferiche P1, P2, P3, P4, ... associate al
35 sistema.

La specifica applicazione nel settore della telefonia cui si è fatto qui riferimento ha peraltro carattere puramente esemplificativo e non implica quindi - neanche in modo indiretto - una limitazione
5 del carattere affatto generale dell'invenzione che sarà descritta nel seguito. Tale invenzione è infatti suscettibile di essere applicata in tutti i campi in cui può essere utile impiegare un microprocessore.

Tornando allo schema della figura 1, la CPU 1 è
10 tipicamente un microprocessore scalare pipelined a 32 bit. Scalare pipelined significa che la sua architettura interna è costituita da diversi stadi logici, ognuno dei quali contiene un'istruzione in un ben specifico stato. Tale stato può essere di:

- 15 - caricamento dell'istruzione stessa dalla memoria,
- decodifica dell'istruzione,
- indirizzamento di un file di registri,
- esecuzione,
- 20 - scrittura/lettura dati dalla memoria.

Il numero di bit su cui opera la CPU 1 si riferisce all'ampiezza dei dati sul quale la macchina opera. Le istruzioni sono generate ed eseguite una alla volta, in uno specifico ordine definito tramite
25 compilazione.

L'altro processore, indicato con DSP, è tipicamente un microprocessore superscalare o VLIW (acronimo per Very Long Instruction Word) pipelined a 128 bit.

30 Superscalare pipelined significa che la sua architettura interna è costituita da diversi stadi logici alcuni dei quali possono eseguire istruzioni in parallelo, ad esempio nella fase di esecuzione. Tipicamente, il parallelismo è di 4 istruzioni ciascuna
35 (pari a 128 bit) mentre i dati sono espressi in 32 bit.

Il processore si dice superscalare se le istruzioni sono riordinate dinamicamente in fase di esecuzione in modo da alimentare gli stadi di esecuzione che potenzialmente possono lavorare in
5 parallelo, anche alterando l'ordine generato staticamente dalla compilazione del codice sorgente, se le istruzioni non presentano dipendenze mutue. Lo svantaggio principale di questo approccio sta nella complessità della macchina risultante, in cui la logica
10 di schedulazione delle istruzioni può risultare essere una delle parti più rilevanti in termini di numero di porte.

Si parla di processore VLIW se le istruzioni sono riordinate staticamente in fase di compilazione ed
15 eseguite in quell'ordine fisso, non modificabile in fase di esecuzione. Il vantaggio di tale approccio è che si elimina tutta la logica di gestione della schedulazione poiché si esegue questo task in sede di compilazione.

20 Lo svantaggio principale sta nel fatto che il codice compilato è strettamente dipendente dall'implementazione della macchina su cui viene eseguito. Ad esempio, pur a parità di architettura di insieme di istruzioni (Instruction Set Architecture o
25 ISA), una macchina con N unità di esecuzione non può eseguire un codice compilato per una macchina con K unità di esecuzione se K non è uguale a N. Da ciò consegue che non c'è "compatibilità binaria" tra diverse generazioni di processori con lo stesso ISA.

30 Si ricorda che per compatibilità binaria si intende la proprietà esistente tra un gruppo di processori ognuno dei quali è in grado di eseguire uno stesso dato codice macchina binario.

Non si possono altresì creare sistemi
35 multiprocessori (ciascuno con diverso numero di unità

di esecuzione) suscettibili di scambiarsi processi in corso di esecuzione.

Nello schema della figura 1, ogni processore possiede la propria cache dati D\$ e la propria cache istruzioni I\$, in modo da poter caricare dalla memoria principale MEM sia i dati sui quali operare, sia le istruzioni da eseguire in parallelo. Poiché i due processori CPU1 e DSP sono collegati alla memoria principale MEM attraverso il bus di sistema B i due processori si trovano tipicamente a competere per l'accesso a tale memoria quando un'istruzione e/o i dati su cui devono operare devono essere localizzati nella memoria principale non essendo disponibili nelle proprie cache.

Un sistema ispirato all'architettura rappresentata nella figura 1 ha una partizione di lavoro e di processi rigida e non modificabile, tale da rendere asimmetrico il carico di lavoro ed i programmi software da eseguire.

A titolo di riferimento, un processore quale la CPU 1 possiede di solito 16 Kbyte di cache e 16 Kbyte di cache istruzioni. Questo mentre il processore DSP possiede di solito 32 Kbyte di cache dati e 32 Kbyte di cache istruzioni.

Il diagramma di flusso della figura 2 illustra lo schema logico della CPU descritto dall'alto verso il basso. Il primo stadio, indicato con 10, genera l'indirizzo di memoria al quale è associato l'istruzione da eseguire, tale indirizzo è detto Program Counter. Lo stadio 10 si configura quindi tipicamente come uno stadio di fetch, mentre l'istruzione così caricata viene decodificata nello stadio 12 separando il campo di bit che ne definisce la funzione (ad esempio di somma di 2 valori contenuti in due registri localizzati nel registro file) rispetto ai

campi di bit che indirizzano gli operandi. Tali indirizzi sono inviati ad un file di registri dal quale (in uno stadio indicato con 14) vengono letti gli operandi dell'istruzione. Gli operandi ed i bit che
5 definiscono la funzione da eseguire sono inviati all'unità di esecuzione che, in uno stadio 16, realizza l'operazione desiderata, ad esempio l'operazione di somma a cui si è fatto riferimento in precedenza. Il risultato può quindi essere ri-memorizzato nel file di
10 registri in uno stadio 18 correntemente denominato stadio di Write Back.

Il processo schematicamente rappresentato dalla figura 2 opera in unione ad un unità di load/store che consente di leggere/scrivere eventuali dati in memoria
15 con l'ausilio di specifiche istruzioni dedicate allo scopo.

E' immediato riconoscere che il set di istruzioni è in corrispondenza biunivoca con una data architettura di CPU di microelaborazione.

20 Il diagramma di flusso della figura 3 mostra invece lo schema logico del processore DSP. Anche in questo caso è previsto uno stadio iniziale di fetch: 20 cui è associato logicamente in cascata uno stadio 20a di emissione delle istruzioni. Il riferimento numerico
25 22 indica invece uno stadio di decodifica mentre il riferimento 24 indica un file registri (si faccia riferimento agli stadi 14 e 16 della figura 2). Il riferimento 28 indica uno stadio di ri-memorizzazione nel file registri complessivamente affine allo stadio
30 18 della figura 1. Nello schema della figura 3 il riferimento 26 indica collettivamente una pluralità di stadi di esecuzione suscettibili di essere eseguiti in parallelo.

Tanto nella figura 1 quanto nella figura 3 il riferimento CW indica le linee di diramazione delle parole di controllo.

Si apprezzerà che la principale differenza fra lo schema della figura 2 e lo schema della figura 3 è data dal fatto che nello schema della figura 3 è prevista la possibilità di lavorare in parallelo su diversi insiemi di istruzioni. Un'altra differenza sta nel fatto che lo schema della figura 3 prevede l'impiego di un maggior numero di unità esecutive disponibili suscettibili di operare in parallelo in un processore superscalare e VLIW. In entrambi i casi il set di istruzioni sta in corrispondenza biunivoca con una data architettura di microelaborazione.

Supponendo che i due insiemi di istruzioni destinati ad essere eseguite dai processori CPU 1 e DSP siano diversi fra loro (così come comunemente si riscontra nell'architettura di processori wireless) si comprende che istruzioni (e quindi task da eseguire) suscettibili di essere eseguiti dal processore CPU 1 non possono essere eseguiti dal processore DSP e viceversa.

Affinché ciò sia possibile è necessario compilare ogni processo per ogni processore, aumentando così la memoria del programma. Ogni volta che un processo deve essere eseguito da uno specifico processore, bisogna poi caricare ed eseguire il codice di quel task che è stato compilato per quel processore. Si riscontra inoltre il problema legato al fatto di dovere correlare i diversi punti di esecuzione parziale dei programmi quando li si voglia spostare da un processore all'altro (ovvero rimappare correttamente i Program Counter), di dovere convertire tutti i dati di elaborazione dal sistema di rappresentazione di un processore all'altro

(ad esempio il contenuto dei registri di stato e general purpose).

Questi problemi sono di difficile risoluzione, per cui in genere un processo viene compilato ed eseguito
5 su un solo processore.

Con riferimento alle figure 4 e 5 è possibile considerare una sequenza di gruppi di istruzioni di detti processi.

In generale si distinguono due tipi di processo,
10 ossia:

- quelli relativi al sistema operativo e ad applicazioni che usano chiamate a funzioni del sistema operativo, e

- quelli relativi all'elaborazione di contenuti
15 multimediali (audio/video/grafica).

In modo specifico, nello schema della figura 4 i riferimenti OsTask 1.1, 1.2, etc. illustrano processi suscettibili di essere eseguiti dal processore CPU1. I processi indicati con MmTask2.1, MmTask2.2, MmTask2.3,
20 ... identificano invece processi compilati in modo da essere eseguiti dal processore DSP.

A partire dallo schema della figura 4, che illustra una possibile assegnazione dei task ai due processori, si può immediatamente risalire allo schema
25 della figura 5 che illustra il corrispondente flusso di istruzioni.

Posto pari a cento il tempo totale di esecuzione dei processi, si osserva che i primi processi tipicamente durano il 10% del tempo, mentre i secondi
30 occupano una parte ben più rilevante, pari al 90%.

Ancora, i primi processi contengono istruzioni generate dal compilatore del processore CPU 1 e quindi possono essere eseguiti dallo stesso, ma non dal processore DSP. Per i secondi processi vale il discorso
35 esattamente complementare, nel senso che contengono

istruzioni generate dal compilatore del processore DSP e possono essere quindi eseguiti da tale processore, ma non dall'altro processore CPU 1.

5 Si osserva ancora che il processore CPU 1 è caratterizzato da un proprio flusso di compilazione, indipendente e distinto da quello del processore DSP.

Dato il modesto carico di lavoro, si evince che il processore CPU 1 potrebbe essere anche spento quando non in uso, consentendo un sensibile risparmio
10 energetico.

Questa ipotetica soluzione (spegnimento del processore CPU 1 quando non utilizzato) si scontra con il fatto che le relative procedure di spegnimento o power-down introducono latenze aggiuntive di
15 elaborazione e si sommano al valore di 10% indicate in precedenza. Le suddette procedure prevedono infatti:

- lo spegnimento del processore CPU1, ad eccezione del rispettivo file registri tramite gating del segnale di clock che alimenta tutti i registri interni,
- 20 - lo spegnimento completo del processore CPU, salvo il fatto che viene mantenuta l'alimentazione energetica delle memorie cache e
- lo spegnimento del CPU nel suo complesso, comprese le cache dati ed istruzioni.

25 Tuttavia, dato che lo stato del singolo processore deve essere ripristinato durante la riaccensione a seguito di una delle operazioni citate in precedenza, le latenze introdotte variano da decine di microsecondi a decine/centinaia di millisecondi. Queste latenze
30 risultano particolarmente dispendiose, sia dal punto di vista energetico, sia dal punto di vista computazionale.

Infine, il processore DSP è costretto a lavorare circa al 90% della sua capacità computazionale. Ciò
35 implica un evidente asimmetria nel carico di lavoro dal

processore CPU rispetto al processore DSP, asimmetria che si rivela anche negli algoritmi di gestione dell'alimentazione (power-management), che sono distinti per i due processori.

5 Scopi e sintesi dell'invenzione

La presente invenzione si prefigge lo scopo di fornire una soluzione in grado di superare gli inconvenienti delineati in precedenza.

10 Secondo la presente invenzione, tale scopo viene raggiunto grazie ad un procedimento avente le caratteristiche richiamate in modo specifico nelle rivendicazioni che seguono. L'invenzione riguarda anche il corrispondente sistema processore, in particolare multiprocessore.

15 La presente invenzione si profila pertanto sia come una variante sia come una possibile integrazione rispetto alla soluzione descritta nella domanda di brevetto europeo No. 01830814.8.

20 In sostanza, la soluzione secondo l'invenzione consente di istanziare una nuova classe di processori con lunghezza di istruzioni selezionabile dal processore stesso. Tutto questo dando origine ad una architettura definibile, con terminologia introdotta nella presente descrizione, Selectable Instruction
25 Length Computer o SILC, dove il parallelismo a livello di istruzioni (ILP o Intruction Level Parallelism) è estratto staticamente dal compilatore.

Tutto questo avviene tuttavia in una forma tale da non pregiudicare la compatibilità binaria fra diversi
30 processori SILC con lo stesso ISA ma parallelismo esecutivo diverso.

Inoltre, il codice può essere eseguito in un numero di cicli ottimale rispetto al parallelismo massimo consentito dal processore.

La soluzione secondo l'invenzione consente dunque di soddisfare in via principale le seguenti esigenze:

- assicurare la compatibilità binaria fra due o più processori SILC aventi lo stesso set di istruzioni
5 ma parallelismo di esecuzione diverso; tutto questo senza utilizzare hardware complesso dedicato allo scopo, così come invece accade nei processori superscalari;
- eseguire programmi in modo indistinto su due o
10 più processori SILC costituenti il sistema in condizioni di lavoro dinamicamente variabile; tali processori sono caratterizzati dal fatto di avere lo stesso set di istruzioni (ISA o Instruction Set Architecture), ma un diverso parallelismo massimo di
15 istruzioni eseguibili in un dato ciclo.

In modo più specifico, la soluzione secondo l'invenzione offre la possibilità di eseguire codice compilato con un compilatore che espone il parallelismo intrinseco a livello di istruzioni di codice,
20 conservando la possibilità di eseguire questo codice su macchine che implementano la stessa architettura di set di istruzioni (ISA) ma con numero diverso di unità di esecuzione. In questo modo, è possibile avere una diversa specifica del numero massimo di istruzioni
25 suscettibili di essere eseguite in un singolo ciclo.

Questo risultato è ottenibile con un processore operante secondo l'invenzione, dunque con lunghezza delle istruzioni selettivamente determinabile, in condizioni in cui il compilatore rivela il massimo
30 parallelismo a livello delle istruzioni inerente nel codice ed inserisce nel codice compilato segni o simboli (token) per identificare istruzioni che - devono - essere eseguite prima di altre nonché istruzioni che - possono - essere eseguite prima di
35 altre.

L'unità di decodifica del processore SILC seleziona, sulla base dei suddetti simboli e del massimo parallelismo della istanza di processore, quante istruzioni possono essere emesse per ciascun
5 ciclo.

In questo modo risulta possibile:

- ottenere la compatibilità binaria tra diverse generazioni di processori che condividono la stessa architettura di set di istruzioni (ISA) ma con diversi
10 implementazioni con numeri di unità e di esecuzione differenti;

- conseguire prestazioni ottimali per ciascun processore con un numero diverso di unità di esecuzione: un processore con un parallelismo di
15 esecuzione più elevato richiederà meno cicli per eseguire lo stesso codice rispetto ad un altro processore con lo stesso ISA, ma meno unità di esecuzione;

- minimizzare la complessità hardware trasferendo
20 la funzione di estrazione del parallelismo al momento della compilazione; e

- disporre di un sistema multiprocessore basato su una schiera di processori SILC che hanno lo stesso valore di ISA, ma che implementano diversi livelli di
25 parallelismo; tutto questo potendo riallocare in modo dinamico i processi da un processore all'altro al fine di minimizzare la frequenza di orologio globale del sistema.

Breve descrizione dei disegni annessi

30 L'invenzione sarà ora descritta, a puro titolo di esempio non limitativo, con riferimento ai disegni annessi, nei quali:

- le figure 1 a 5, sostanzialmente relative alla tecnica nota, sono già state descritte in precedenza;

- la figura 6 illustra il principio di funzionamento di un processore secondo l'invenzione, la spiegazione proseguendo nelle figure 7 a 13,

- la figura 14 illustra una architettura
5 multiprocessore suscettibile di implementare l'invenzione,

- la figura 15 illustra i criteri di assegnazione dei task nel sistema multiprocessore della figura 14,

- le figure 16 e 17 illustrano in maggior
10 dettaglio i criteri di esecuzione dei suddetti task nel processore della figura 14, e

- la figura 18 è un esempio di tabella di esecuzione di processi riferito ad una soluzione secondo l'invenzione.

15 Un processore secondo l'invenzione (nel seguito anche indicato in breve con l'acronimo SILC) è suscettibile di essere utilizzato in modo particolarmente vantaggioso nell'ambito di una struttura multiprocessore del tipo di quella
20 rappresentata nella figura 14, sulla quale si tornerà nel seguito.

In sostanza, un processore secondo l'invenzione presenta la caratteristica data dal fatto che, così come un processore VLIW, esso sfrutta il compilatore
25 per estrarre il parallelismo a livello di istruzioni e ridurre in questo modo la complessità realizzativa del hardware.

A differenza di quanto avviene in un normale processore VLIW, in un processore secondo l'invenzione
30 il codice compilato non è eseguibile solo su un processore con un dato parallelismo come presupposto dal compilatore, ma può essere eseguito da diversi processori SILC caratterizzati da uno stesso ISA ma con un livello di parallelismo diverso. Tutto questo
35 ottenendo prestazioni che aumentano all'aumentare del

numero massimo di istruzioni che un processore esegue in un ciclo.

Un compilatore VLIW deve infatti conoscere esattamente quante (e quali) unità di esecuzione ha il processore per cui compila il codice. Questo al fine di generare il numero di istruzioni che il processore deve svolgere ad ogni ciclo. Se, ad esempio, si specifica un processore a parallelismo 4, il compilatore cerca di far eseguire sino a quattro istruzioni per ciclo. Se il codice viene eseguito su un processore con parallelismo 8, quattro unità rimangono inutilizzate. Viceversa un tale codice non può essere eseguito su un processore con due sole unità di esecuzione.

Un processore superscalare utilizza invece risorse hardware dedicate (che possono essere molto costose) per capire quante istruzioni può eseguire.

Di conseguenza, la soluzione secondo l'invenzione mantiene la flessibilità di un processore superscalare, eliminando però la complessità hardware aggiuntiva.

Il flusso di compilazione-esecuzione di un processore SILC prevede i seguenti passi.

In primo luogo, il compilatore riceve come ingresso un programma scritto, ad esempio, in codice C, o in qualsiasi altro linguaggio ad alto livello od anche in assembler.

Successivamente, il compilatore traduce tale programma in istruzioni native del processore. Mentre effettua la traduzione, il compilatore estrae anche il massimo (o alternativamente, fino ad un massimo stabilito a piacere di istruzioni per ciclo) parallelismo a livello di istruzioni assembler intrinseco nel codice, ovvero indica quali istruzioni possono essere eseguite in parallelo senza che ciò provochi una variazione del comportamento del programma

dovuto a dipendenze di dati da un'istruzione ad un'altra.

Ogni gruppo di istruzioni suscettibile di essere eseguite in parallelo è definito "bundle".

5 Nel fare ciò, il compilatore non assume alcunché di specifico per quanto riguarda il numero di unità di esecuzione che ha a disposizione il processore su cui si esegue il codice. Il compilatore estrae tutto il parallelismo possibile insito nel codice, oppure fino
10 ad un dato numero di istruzioni al ciclo che si può ritenere ragionevole, secondo le opzioni che possono venire date all'ingresso del compilatore.

In ogni bundle si possono in genere distinguere due categorie di istruzioni. La prima di tali categorie
15 comprende le istruzioni che - devono - essere necessariamente eseguite prima di quelle appartenenti al bundle successivo a causa del fatto che queste ultime ricevono in ingresso dati elaborati dalle prime: questo primo tipo di istruzioni può essere definito
20 "must". L'altra categoria di istruzioni consiste in istruzioni che - possono - essere eseguite sia prima del prossimo bundle, sia parallelamente al prossimo bundle, in quanto non hanno dipendenze specifiche con istruzioni presenti nel prossimo bundle: questo secondo
25 tipo di istruzioni può essere definito di tipo "can".

In ogni bundle può essere presente un qualsiasi insieme (compreso l'insieme vuoto, dunque comprende zero istruzioni) di ognuna delle due categorie.

Ad esempio, lo schema della figura 6 fa vedere
30 come un codice in C (rappresentato a sinistra) è compilato nell'insieme di istruzioni native del processore SILC, estraendo il valore di parallelismo massimo, ovvero il valore massimo di tale parallelismo fino ad un limite superiore predefinito, sulla base
35 delle indicazioni fornite al compilatore. Il suddetto

valore massimo viene correntemente indicato come ILP. Le istruzioni suscettibili di essere eseguite in parallelo vengono raggruppate nei suddetti "bundle".

Così come meglio schematizzato nella figura 7, 5 ciascun gruppo di istruzioni denominato bundle è in generale costituito tanto da istruzioni che - devono - essere eseguite necessariamente prima dell'esecuzione del bundle successivo quanto da istruzioni che - possono - essere eseguite prima del bundle successivo, 10 ovvero in parallelo con il bundle successivo.

Di conseguenza, le istruzioni di assembler possono essere etichettate come "must" o "can" a seconda della categoria in cui si trova la rispettiva istruzione. Come si è già detto, ciascun bundle può contenere un 15 numero qualsiasi, (compreso lo zero) di istruzioni da ciascun gruppo.

Come conseguenza della partizione descritta è possibile definire una sequenza di sottogruppi o sub-bundle di tipo "must" ovvero di tipo "can" (ognuno 20 contenente zero o più istruzioni) destinati ad alternarsi nell'esecuzione del programma, così come rappresentato nella figura 8.

A questo punto, un qualsiasi processore SILC che esegue istruzioni dall'insieme di istruzioni utilizzato 25 dal compilatore può eseguire il codice compilato e può selezionare di volta in volta automaticamente, attraverso le indicazioni di sub-bundle must/can dati dal compilatore, la lunghezza di istruzioni da eseguire ciclo per ciclo, ovvero il numero di istruzioni 30 assembler da eseguire in ogni ciclo.

Il processore seleziona tale lunghezza a partire da un determinato insieme di regole, quali, tipicamente:

- il processore cerca di caricare sempre il 35 massimo numero di istruzioni che può eseguire,

- il processore può eseguire tutte le istruzioni caricate se e soltanto se non sono presenti istruzioni "must" appartenenti a bundle diversi (situazione di conflitto o hazard must-must); in tal caso può eseguire
5 solo le istruzioni fino alla prima corrispondente al secondo sub-bundle di tipo "must", quest'ultimo escluso. Dovrà invece ritardare l'esecuzione delle istruzioni "must" del secondo bundle al ciclo successivo.

10 Gli schemi delle figure 9 a 11 fanno vedere tipici esempi di "mixture" di tipi di bundle diversi ed esempi della corrispondente frequenza di esecuzione di codice su processori a parallelismo 4 (figure 9 e 10) e parallelismo 8 (figura 11).

15 In particolare, la figura 9 fa riferimento ad un generico flusso di 4 sub-bundle 0M, 0C, 1M e 1C, dove, evidentemente, M sta per "must" e C per "can".

In particolare, procedendo dall'alto si vede che i primi tre esempi portano a configurazioni per cui è
20 possibile procedere immediatamente con l'esecuzione.

Al contrario, la quarta combinazione rappresentata, in cui è presente un sub-bundle 0M, un sub-bundle 0C ed un sub-bundle 1M non può ricevere immediatamente la concessione di esecuzione, in quanto
25 insorge un fenomeno di conflitto o hazard di tipo must-must.

In questo caso si opera con una scissione o split allocando l'insieme sub-bundle 0M, sub-bundle 0CX e l'insieme sub-bundle 1MX su due cicli diversi. Il tutto
30 con la possibilità di aggiungere, in fase di esecuzione del secondo ciclo, altre istruzioni di tipo "must" o "can" dello stesso bundle 1.

La figura 10 illustra invece un esempio di esecuzione del codice compilato illustrato nelle figure

7 e 8 su una macchina con 4 unità di esecuzione, dunque con parallelismo 4.

Il funzionamento illustrato prevede un'esecuzione su sette cicli.

5 La figura 11 si riferisce invece ad un esempio di esecuzione dello stesso codice compilato illustrato nelle figure 7 e 8 su una macchina a parallelismo 8, vale a dire con otto unità di esecuzione.

10 In questo caso, l'esecuzione complessiva del codice prevede quattro cicli. Durante i primi due cicli non è possibile eseguire più istruzioni rispetto a quelle rappresentate. Questo in quanto l'esecuzione in parallelo di istruzioni "must" appartenenti a bundle diversi è proibita come evento di conflitto must-must.

15 Si può quindi notare che lo stesso codice - compilato senza far riferimento ad un parallelismo di esecuzione particolare - può essere eseguito su macchine con parallelismo diverso, ottenendo prestazioni che naturalmente scalano con il numero di
20 unità di esecuzione in parallelo: evidentemente, l'aumentare del numero di unità di esecuzione disponibili riduce il numero di cicli necessari all'esecuzione.

25 L'informazione relativa ai diversi tipi di sub-bundle di tipo must o can può essere codificata secondo criteri diversi.

Alcuni possibili criteri vengono qui sotto elencati a titolo di esempio.

30 Così come illustrato nella figura 12, si può assegnare ad un bit dell'istruzione il significato "must_not_can". In questo modo, tutte le istruzioni appartenenti al sub-bundle must avranno tale bit a 1. Tutte le istruzioni appartenenti al sub-bundle "can" hanno tale bit a 0. Il passaggio da un sub-bundle ad un

altro è quindi identificato da un inversione di tale bit.

Nel caso in cui il numero di istruzioni appartenenti ad un sub-bundle "can" sia nullo, si rende
5 necessario aggiungere un istruzione fittizia "can" di tipo no-operation (nop) per significare il passaggio da un gruppo must ad un altro. Lo stesso avviene nel caso opposto di due gruppi can intercalati da un gruppo must formato da 0 istruzioni (anche se questo caso appare in
10 realtà difficile da ipotizzarsi).

In alternativa si può assegnare a due distinti bit il significato di "ultima istruzione di un sottogruppo must" e "ultima istruzione di un sottogruppo can". Questa soluzione è schematicamente illustrata nella
15 figura 13.

In questo modo si utilizza un bit in più, ma si elimina la necessità di introdurre istruzioni di tipo no-operation (nop) ridondanti in caso di presenza di un sub-bundle a 0 istruzioni.

20 La soluzione secondo l'invenzione fa sì che per supportare l'esecuzione di processi in un contesto quale quello in precedenza illustrato con riferimento alla figura 1, non sia più necessario l'uso di un'architettura multi-processing asimmetrica. Questo in
25 quanto si può stanziare un processore SILC a basso parallelismo per le istruzioni svolte normalmente dalla CPU ed un processore SILC ad alto parallelismo per le funzioni che, nello schema della figura 1, sono svolte dal processore DSP.

30 E' allora conveniente definire una nuova architettura di sistema, del tipo di quella illustrata nella figura 14, dove parti identiche ovvero funzionalmente equivalenti a quelle illustrate nella figura 1 sono state indicate con gli stessi
35 riferimenti.

In particolare, il posto occupato nella figura 1 dai processori CPU1 e DSP è occupato nello schema della figura 14 da due processori secondo l'invenzione indicati rispettivamente con SILC 1 e SILC 2.

5 L'architettura della figura 14 consente di eseguire i processi su entrambe i processori senza per questo dover ricompilare e duplicare i codici oggetto per i due processori. Questo è possibile poiché entrambi i processori SILC1 e SILC2 supportano lo
10 stesso instruction set e lo stesso codice binario può essere eseguito da macchine che hanno un diverso livello di parallelismo esecutivo.

In particolare, con riferimento alle figure 15 e 16 si considera una prima fase di compilare il codice sorgente di un processo detto OSTask 1.1 relativo al
15 sistema operativo, con il compilatore SILC. Tale codice è in genere caratterizzato da un basso valore intrinseco del parametro ILP (si faccia riferimento alla descrizione della figura 6) e può essere eseguito
20 da entrambi i processori. E' però evidente che, se eseguito su un processore con elevato parallelismo massimo, il tempo di esecuzione non si riduce molto rispetto alla situazione che insorge se si utilizza un processore a basso parallelismo, a causa del basso ILP
25 intrinseco. Tale task è quindi più efficientemente eseguito sul processore SILC 1.

Inoltre, si consideri nella stessa detta prima fase di compilare il codice sorgente di un processo detto MmTask 2.1 relativo ad un'applicazione
30 audio/video/grafica multimediale, con il compilatore SILC. Tale codice è in genere caratterizzato da un alto ILP intrinseco e, come sopra, può essere nominalmente eseguito da entrambi i processori. E' però evidente che, se eseguito su un processore con elevato
35 parallelismo massimo, il tempo di esecuzione si riduce

rispetto alla situazione che insorge se si utilizza un processore a basso parallelismo. Tale task è quindi più efficientemente eseguito sul processore SILC 2.

La figura 16 esemplifica quanto detto sopra nel
5 caso in cui SILC 1 abbia parallelismo 4 e SILC 2 abbia parallelismo 8.

Le istruzioni che compongono i task di figura 15 sono presenti nella memoria del sistema e sono indirizzate mediante il Program Counter che ogni SILC
10 possiede (si veda lo schema della figura 17).

Un vantaggio rilevante della soluzione secondo l'invenzione è la compatibilità binaria fra i due processori, intendendosi tale compatibilità come la capacità di poter eseguire indistintamente i processi
15 sui processori coinvolti con diverso massimo parallelismo esecutivo, usando lo stesso codice compilato senza inutili duplicazioni. Questa capacità consente inoltre di distribuire dinamicamente il carico computazionale sui due processori in modo da
20 equalizzare la frequenza operativa dei processori rispetto al massimo ottenendo così un risparmio di potenza dissipata che, come ben noto, dipende linearmente dalla frequenza operativa del processore.

Per meglio chiarire come sia possibile spostare
25 l'esecuzione di un processo da un processore SILC all'altro si consideri l'esistenza di una tabella memorizzata nella memoria MEM del sistema.

Con riferimento alla figura 18a, procedendo da sinistra verso destra, tale tabella è composta da:

- 30 - una lista di processi in esecuzione o sospesi su un qualsivoglia processore (Process),
 - il numero progressivo (Num) dello stesso in base all'ordine di attivazione,
 - la percentuale di potenza massima (CPU load) del
35 processore che è utilizzata da detto processo,

- il tempo di esecuzione (Exec. Time),
- la quantità di memoria (Memory) del sistema usata da detto processo per poter eseguire la funzione alla quale è preposto,

5 - il processore su cui il processo attualmente risiede (Current execution) e,

- l'indirizzo della porzione di memoria nella quale i dati e le istruzioni sono memorizzati ossia il contesto operativo (Context Memory)).

10 Questa tabella è accessibile da un processo, detto di controllo, che viene eseguito per un tempo prefissato su uno dei processori. Tale processo ha la possibilità di consultare ed aggiornare la tabella al fine di equalizzare il carico di lavoro del rispettivo
15 processore rispetto al carico di lavoro dell'altro o degli altri processori presenti nel sistema.

La soluzione secondo l'invenzione è infatti estendibile ad un numero arbitrario di processori SILC che costituiscono un sistema ed ognuno dei quali abbia
20 un qualunque parallelismo esecutivo massimo, dove tale lunghezza vari da un processore all'altro.

Tale tabella contiene altresì le coordinate necessarie affinché un processore possa prendere possesso ed eseguire uno dei processi citati.

25 Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno essere ampiamente variati rispetto a quanto descritto ed illustrato, senza per questo uscire dall'ambito della presente invenzione,
30 così come definito dalle rivendicazioni annesse.

RIVENDICAZIONI

1. Procedimento per eseguire programmi su almeno un processore (SILC 1, SILC 2) avente un'architettura di insieme di istruzioni data (ISA), caratterizzato dal fatto che comprende le operazioni di:

- compilare il programma da eseguire traducendolo in istruzioni native di detta architettura di insieme di istruzioni (ISA), organizzando le istruzioni derivanti dalla traduzione di detto programma in
10 rispettivi gruppi disposti in un ordine di gruppi successivi, ciascun gruppo raggruppando istruzioni suscettibili di essere eseguite in parallelo da detto almeno un processore,

- ordinare detti gruppi di istruzioni in
15 rispettivi sottogruppi, detti sottogruppi identificando un primo insieme di istruzioni (0M, 1M, 2M, 3M, ...) che devono essere eseguite prima delle istruzioni appartenenti al gruppo successivo in detto ordine ed un secondo insieme di istruzioni (0C, 1C, 2C, 3C, etc.)
20 che possono essere eseguite sia prima sia in parallelo rispetto alle istruzioni appartenenti a detto gruppo successivo in detto ordine; almeno detto secondo insieme di istruzioni potendo essere l'insieme vuoto,

- definire una sequenza di esecuzione delle
25 istruzioni di detti sottogruppi in successivi cicli di funzionamento di detto almeno un processore (SILC 1, SILC 2) evitando, nell'assegnare ciascun sottogruppo ad un ciclo di funzionamento del processore, l'assegnazione simultanea allo stesso ciclo di
30 funzionamento di due sottogruppi (0M, 1M) corrispondenti ad istruzioni appartenenti a detto primo insieme di due gruppi successivi in detto ordine, e

- eseguire dette istruzioni su detto almeno un processore (SILC 1, SILC 2) rispettando detta sequenza
35 di esecuzione.

2. Procedimento secondo la rivendicazione 1, caratterizzato dal fatto che comprende l'operazione di variare selettivamente la lunghezza complessiva di istruzione eseguita per ogni ciclo da detto almeno un
5 processore (SILC 1, SILC 2).

3. Procedimento secondo la rivendicazione 1 o la rivendicazione 2, caratterizzato dal fatto che comprende l'operazione di identificare le istruzioni appartenenti ad un sottogruppo di detto primo (0M, 1M,
10 2M, 3M ...) ed a detto secondo (0C, 1C, 2C, 3C, ...) insieme tramite un simbolo binario posto rispettivamente ad un primo e ad un secondo valore logico.

4. Procedimento secondo la rivendicazione 3,
15 caratterizzato dal fatto che comprende l'operazione di:

- rilevare quando uno fra detto primo e detto secondo insieme è l'insieme vuoto, e
- inserire nel rispettivo sottogruppo un'istruzione fittizia che non implica lo svolgimento
20 di operazioni (no-operation).

5. Procedimento secondo la rivendicazione 1 o la rivendicazione 2, caratterizzato dal fatto che comprende l'operazione di identificare le istruzioni appartenenti ad un sottogruppo di detto primo (0M, 1M,
25 2M, 3M, ...) ed a detto secondo (0C, 1C, 2C, 3C, ...) insieme tramite due distinti simboli binari identificativi dell'ultima istruzione del rispettivo sottogruppo.

6. Procedimento secondo una qualsiasi delle
30 rivendicazioni 1 a 5, per eseguire programmi su un sistema multiprocessore comprendente una pluralità di processori (SILC 1, SILC 2, ...) aventi detta architettura di insieme di istruzioni (ISA), caratterizzato dal fatto che comprende le operazioni
35 di:

- istanziare i processori (SILC 1, SILC 2, ...) di detta pluralità con rispettivi gradi di parallelismo di esecuzione con almeno due valori diversi di detto parallelismo di esecuzione nell'ambito di detta pluralità, e

- distribuire selettivamente l'esecuzione delle istruzioni di detta sequenza di esecuzione fra i processori di detta pluralità (SILC 1, SILC 2, ...), le istruzioni di detta sequenza di esecuzione essendo
10 direttamente eseguibili da parte dei processori di detta pluralità (SILC 1, SILC 2) in condizioni di compatibilità binaria.

7. Procedimento secondo la rivendicazione 6, caratterizzato dal fatto che comprende l'operazione di
15 distribuire selettivamente l'esecuzione delle istruzioni di detta sequenza tra i processori di detta pluralità (SILC 1, SILC2) distribuendo dinamicamente il carico computazionale di detti processori.

8. Procedimento secondo la rivendicazione 7,
20 caratterizzato dal fatto che comprende l'operazione di distribuire selettivamente l'esecuzione delle istruzioni di detta sequenza fra detti processori di detta pluralità (SILC 1, SILC 2) con il criterio di equalizzare la frequenza operativa dei processori di
25 detta pluralità (SILC 1, SILC 2, ...).

9. Procedimento secondo una qualsiasi delle rivendicazioni 6 a 8, caratterizzato dal fatto che comprende l'operazione di svolgere un processo di controllo eseguito da almeno uno dei processori di
30 detta pluralità così da equalizzare il proprio carico di lavoro rispetto agli altri processori di detto sistema multiprocessore.

10. Procedimento secondo la rivendicazione 9, caratterizzato dal fatto che comprende l'operazione di
35 costituire una tabella accessibile da detto processo di

controllo, detta tabella comprendendo voci scelte nel gruppo costituito da:

- una lista di processi (Process) in esecuzione o sospesi su un qualsivoglia processore di detta
5 pluralità,

- il numero progressivo (Num) dello stesso in base all'ordine di attivazione,

- la percentuale di potenza massima (CPU load) del processore che è utilizzata da detto processo,

10 - il tempo di esecuzione (Exec. Time),

- la quantità di memoria (Memory) del sistema usata da detto processo per poter eseguire la funzione alla quale è preposto,

- il processore su cui il processo attualmente
15 risiede (Current execution) e,

- l'indirizzo della porzione di memoria nella quale i dati e le istruzioni sono memorizzati (Context Memory)).

11. Sistema processore, preferibilmente di tipo
20 multiprocessore, configurato per operare con il procedimento secondo una qualsiasi delle rivendicazioni

1 a 10.

RIASSUNTO

Il programma da eseguire viene compilato traducendolo in istruzioni native dell'architettura di insieme di istruzioni (ISA) del sistema processore (SILC1, SILC2), organizzando le istruzioni derivanti dalla traduzione del programma in rispettivi gruppi disposti in un ordine di gruppi successivi, ciascun gruppo raggruppando istruzioni suscettibili di essere eseguite in parallelo dal sistema un processore. I gruppi di istruzioni vengono ordinati in rispettivi sottogruppi, che identificando un primo insieme di istruzioni ("must") che devono essere eseguite prima delle istruzioni appartenenti al gruppo successivo in detto ordine ed un secondo insieme di istruzioni ("can") che possono essere eseguite sia prima sia in parallelo rispetto alle istruzioni appartenenti a detto gruppo successivo in detto ordine. Si definisce una sequenza di esecuzione delle istruzioni in successivi cicli di funzionamento del sistema processore (SILC 1, SILC 2) assegnando ciascun sottogruppo ad un ciclo di funzionamento, evitando l'assegnazione simultanea allo stesso ciclo di funzionamento di due sottogruppi appartenenti al primo insieme ("must") di due gruppi successivi. Le istruzioni della sequenza sono eseguibili da parte dei vari processori del sistema (SILC1, SILC2) in condizioni di compatibilità binaria.

(Figura 14)

Fig. 1

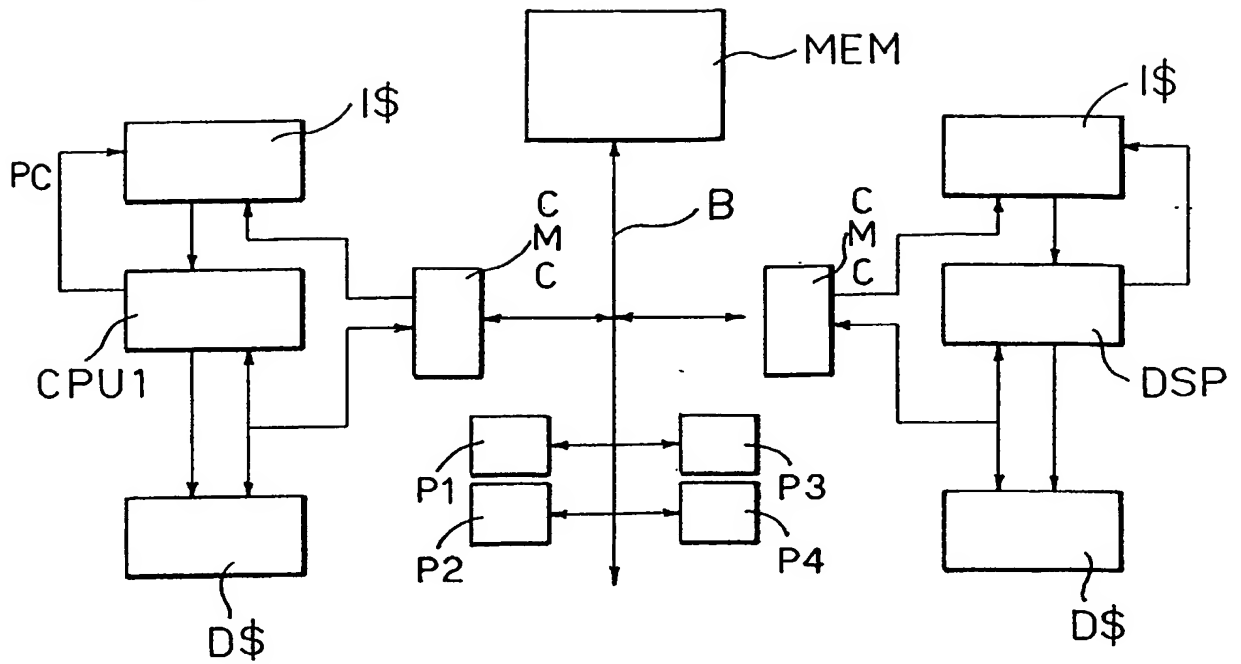


Fig. 2

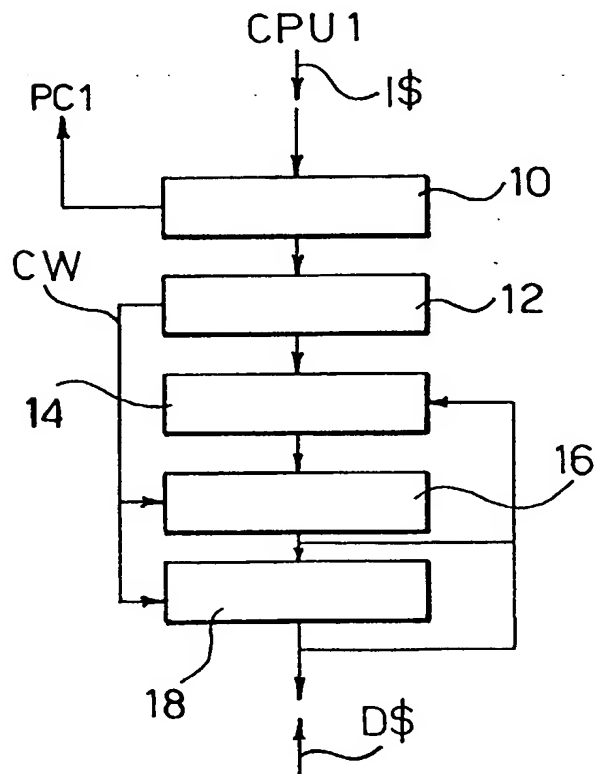


Fig. 3

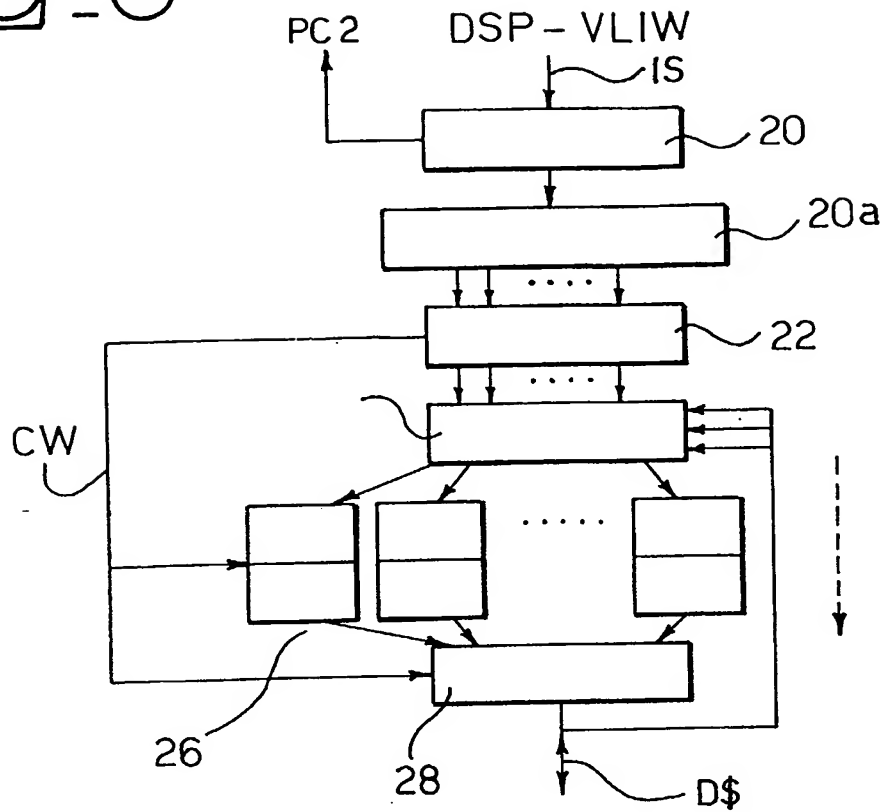
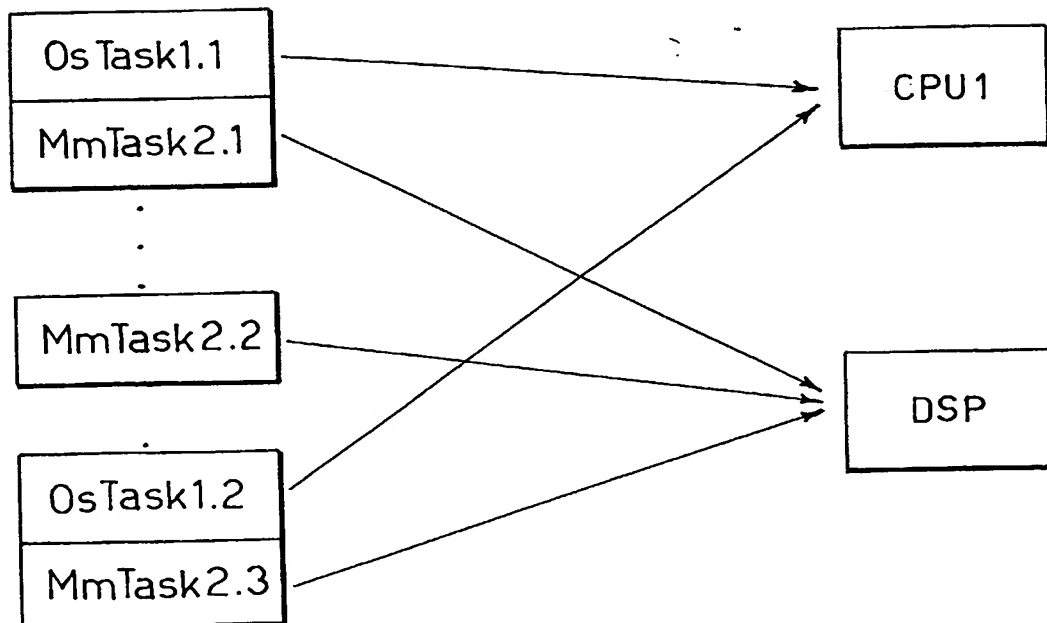
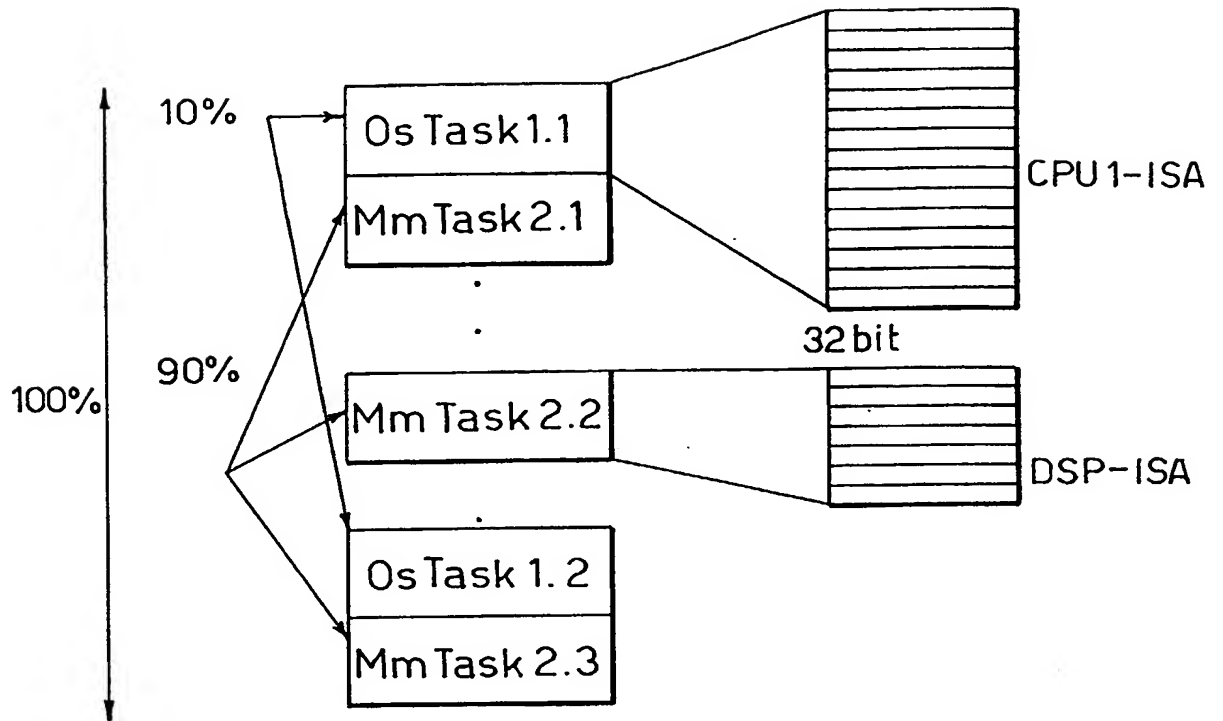


Fig. 4



Fig_5



Fig_6

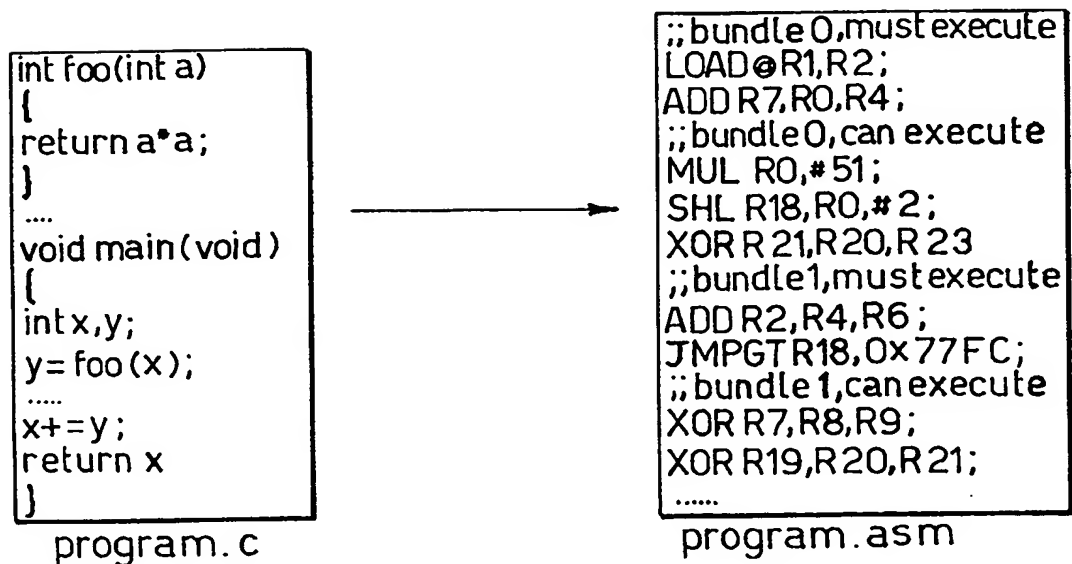


Fig. 7

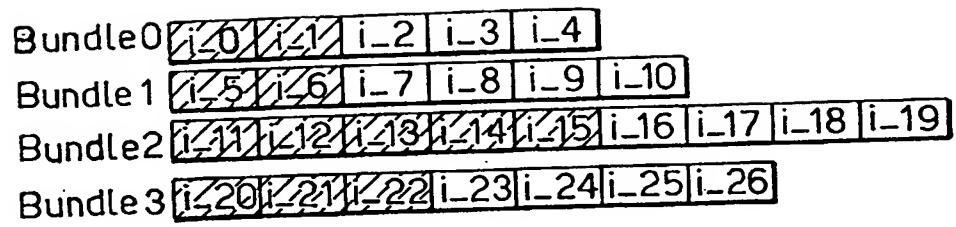


Fig. 8

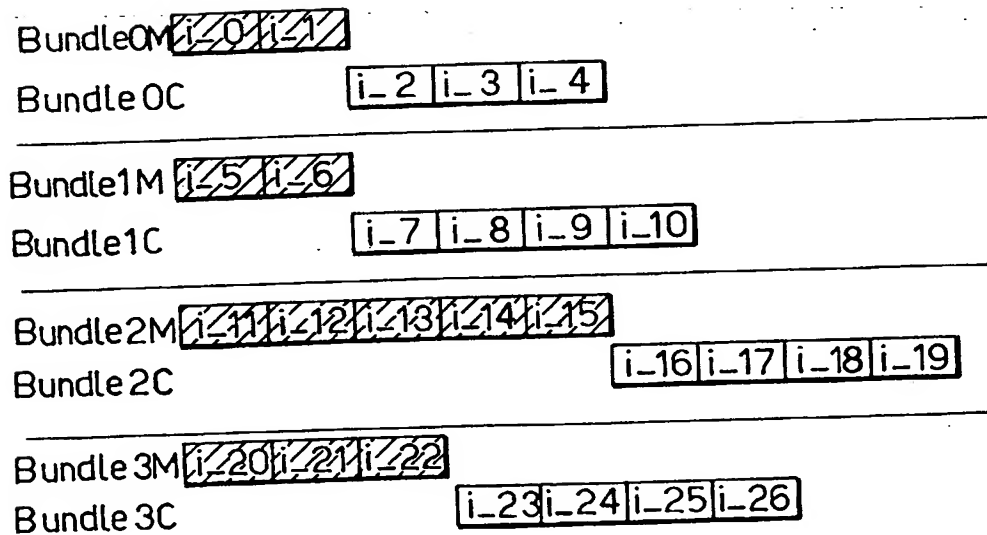



Fig 9

 Sub-Bundle OM, sub-Bundle OC: allowed

 Sub-Bundle OC, sub-Bundle 1M, sub-Bundle 1C: allowed

 Sub-Bundle OC, sub-Bundle 1M: allowed

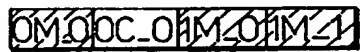
 Sub-Bundle OM, sub-Bundle OC, sub-Bundle 1M: NOT ALLOWED

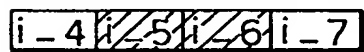






Fig 10

 Bundle OM, Bundle OC

 Bundle OC, Bundle 1M, Bundle 1C

 Bundle 1C, Bundle 2M

 Bundle 2M

 Bundle 2C

 Bundle 3M, Bundle 3C

 Bundle 3C, etc....

Fig - 13

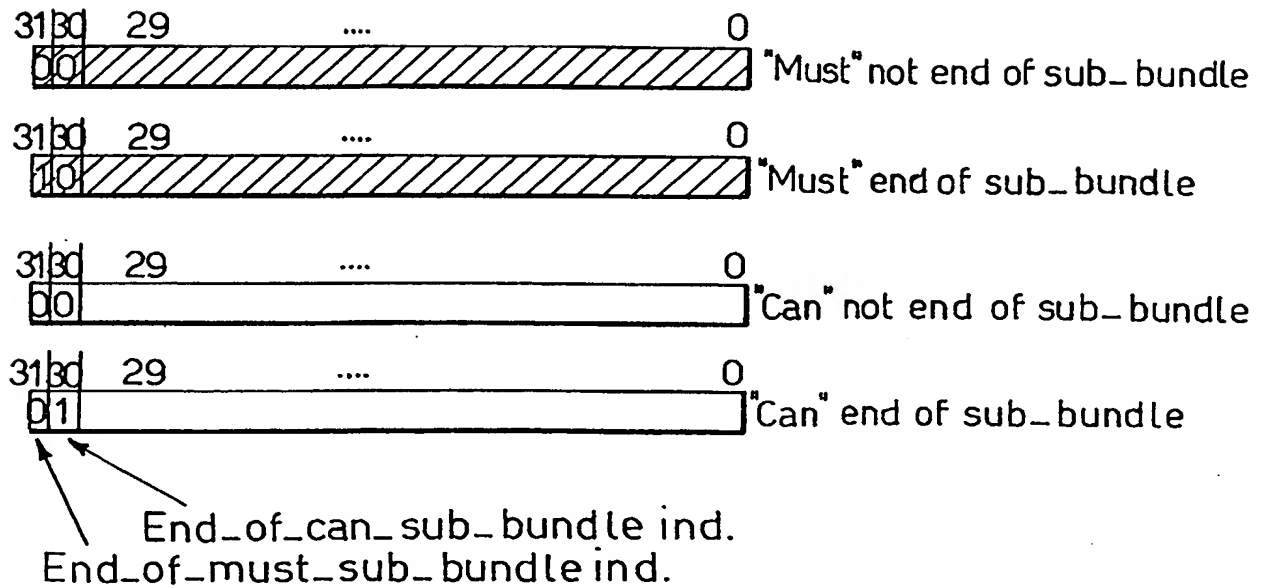


Fig - 14

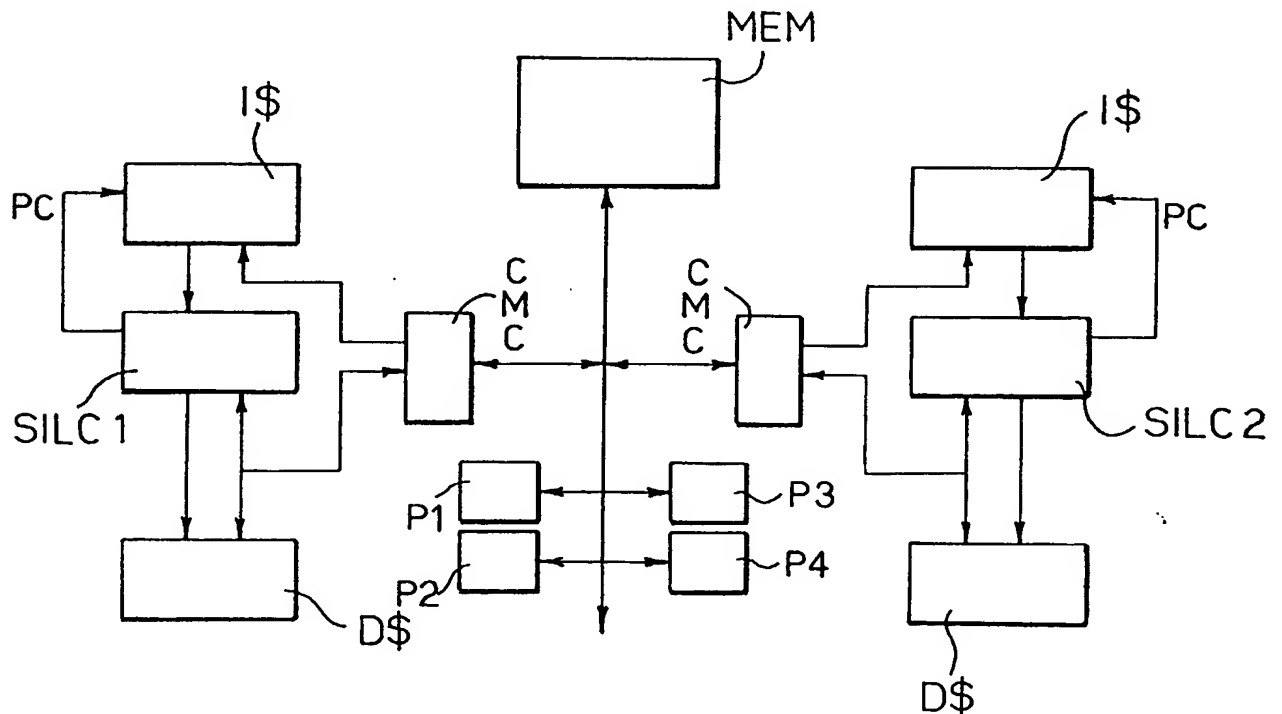


Fig. 15

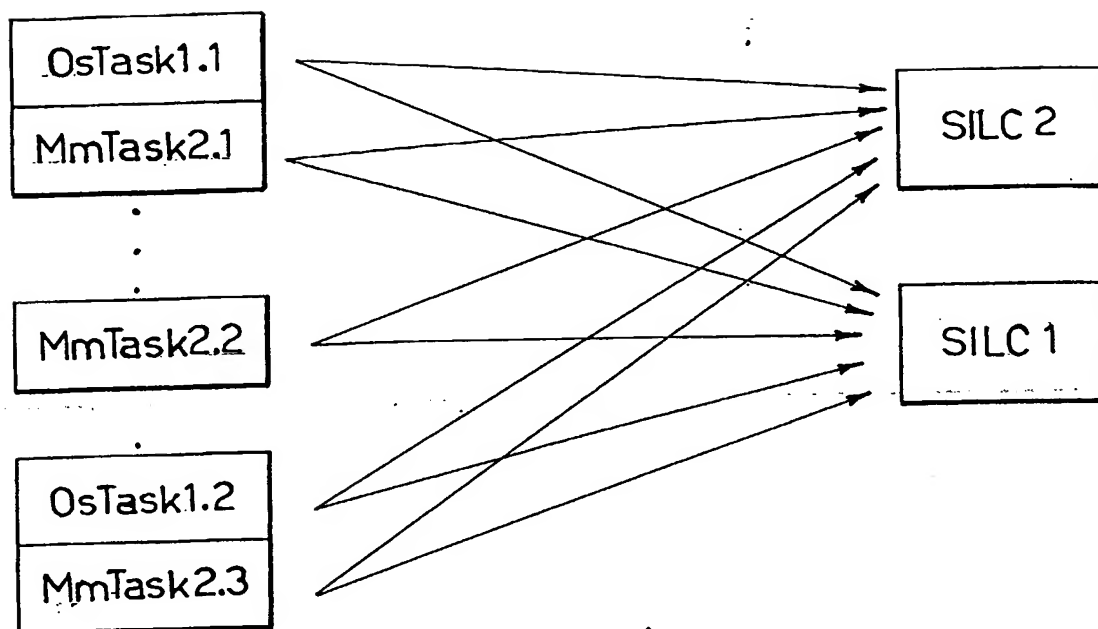


Fig. 16

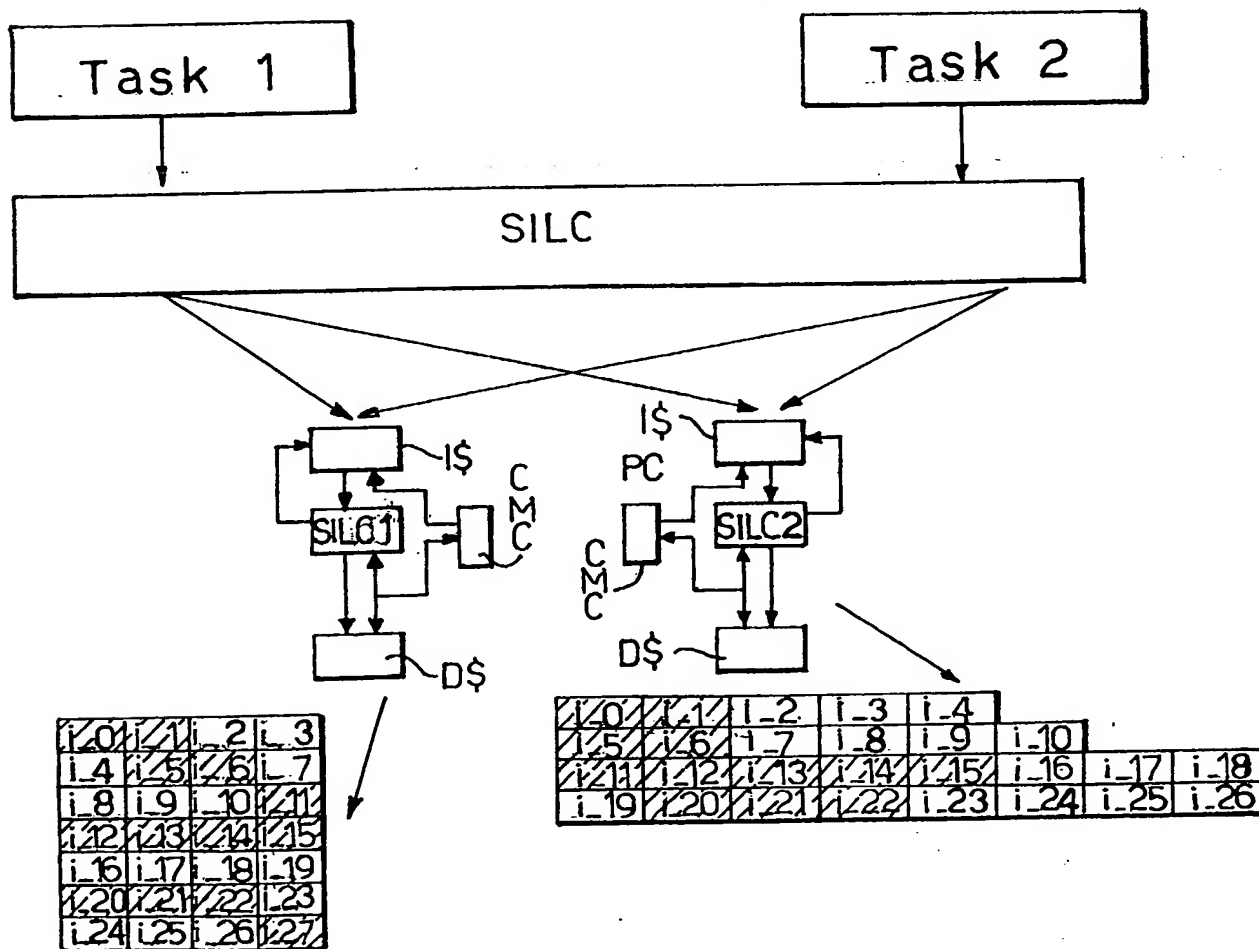


Fig - 17

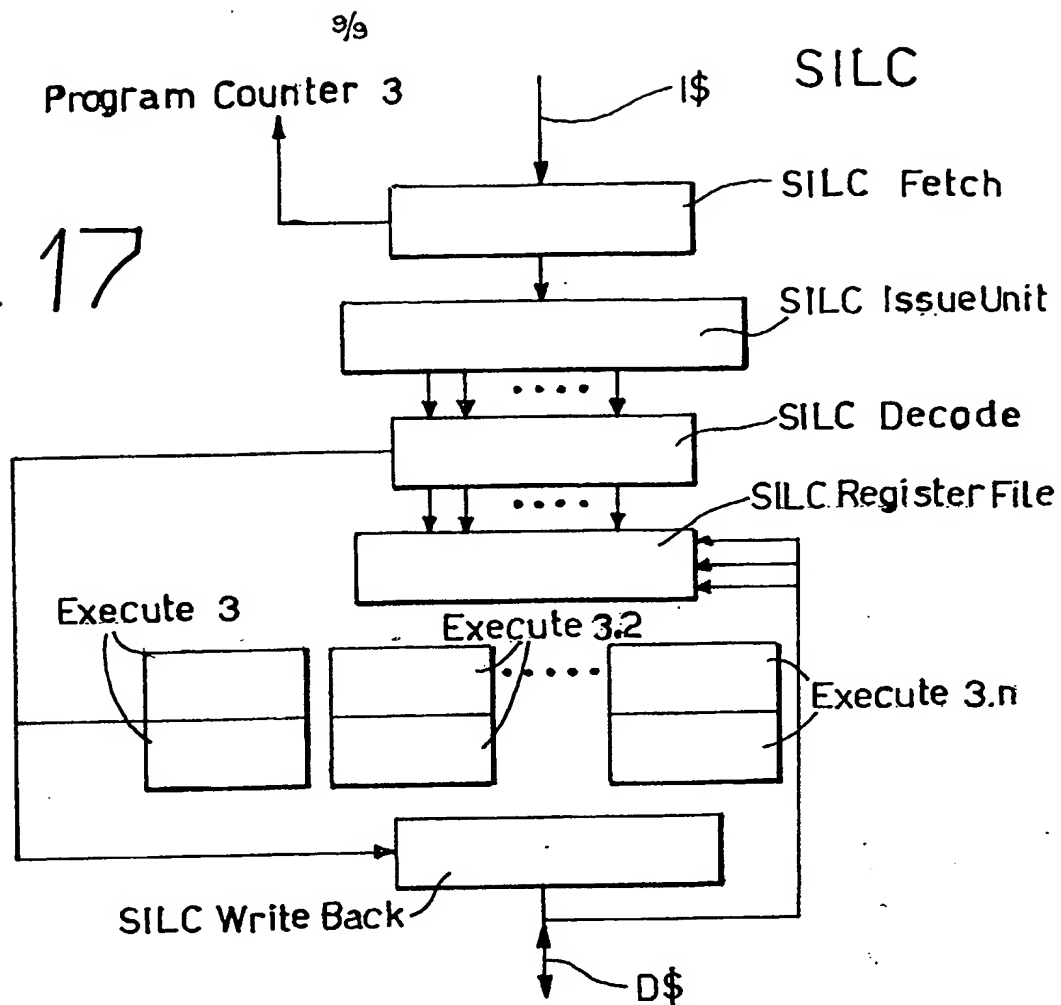


Fig - 18

| Process | Num. | CPU load | Exec time | Memory | Current execution | Context memory address |
|-------------------|------|----------|-----------|--------|-------------------|------------------------|
| SystemIdleProcess | 0 | 52 | 4:18:39 | 16K | 1 | 0x1234123A |
| System | 2 | 02 | 0:00:37 | 200K | 2 | 0x3E586321 |
| SMSS.EXE | 26 | 00 | 0:00:00 | 0K | 2 | 0xFFC0A867 |
| CSRSS.EXE | 34 | 00 | 0:00:06 | 908K | 1 | 0xAFE25432 |
| WINLOGON.EXE | 40 | 00 | 0:00:01 | 32K | 1 | 0xCECC4223 |
| SERVICES.EXE | 43 | 00 | 0:00:01 | 1044K | 1 | |

